# IMDB Movie Recommender System:

# Content-Based Filtering Implementation and Analysis

Shashank Keshary Kansas State University CIS 731

## Abstract

In this project, I implemented a content-based movie recommender system utilizing the IMDB dataset. Through data quality filtering and efficient feature engineering, I process over 500,000 movies to create a system that provides explainable recommendations. My implementation successfully processes 26,107 quality-filtered movies and demonstrates high recommendation accuracy with comprehensive explanation capabilities. The system achieves this while maintaining computational efficiency through careful memory optimization and feature selection techniques.

## 1. Introduction

The challenge of providing personalized movie recommendations has grown significantly with the expansion of digital streaming platforms and online content. In my project, I address this challenge through the implementation of a content-based recommender system that leverages comprehensive movie metadata from IMDB. My approach focuses on balancing recommendation quality with computational efficiency, addressing the key challenges of processing large-scale data while maintaining the ability to provide meaningful, explainable recommendations to users.

The scope of my implementation incorporates several critical areas of data science. I begin with the processing of raw IMDB dataset files, progress through feature engineering, and culminate in the development of a memory-optimized recommendation engine. Throughout the implementation, I maintain a focus on data quality and processing efficiency, ensuring that my system can handle large-scale data while providing high-quality recommendations.

## 2. Dataset and Processing

The foundation of my implementation rests on the IMDB dataset, which provides comprehensive movie information through multiple structured files. My system primarily utilizes three core dataset files: title.basics.tsv for fundamental movie information, title.ratings.tsv for user rating data, and title.crew.tsv for director and crew information. The initial dataset, containing over 500,000 movies, undergoes careful quality filtering to ensure recommendation reliability.

My data processing pipeline implements a sophisticated approach to quality control and feature preparation. I begin by selectively loading relevant columns to optimize memory usage from the start. I then apply a series of quality filters, including a minimum threshold of 1,000 user votes and requirements for complete genre and director information. This filtering process results in a refined dataset of 26,107 movies, each with comprehensive metadata suitable for generating reliable recommendations.

The feature engineering process I developed transforms this raw data into a format suitable for content-based recommendation. I implement text vectorization for genre processing, creating efficient representations of movie features. I process director information through custom tokenization, and normalize release years to maintain temporal relevance in recommendations. This careful preparation of features ensures that my similarity calculations capture meaningful relationships between.

## 3. Technical Implementation

The core of my recommender system is implemented through a carefully structured Python class that manages all aspects of data processing and recommendation generation. My implementation emphasizes memory efficiency and processing speed while maintaining recommendation quality. The following code excerpt demonstrates the foundational structure of my system:

class IMDBRecommender:  
 def \_\_init\_\_(self):  
 self.movies\_df = None   
 self.feature\_matrix = None   
 self.similarity\_matrix = None   
 self.vectorizer = None   
  
 def process\_data(self):  
 """Loads and processes IMDB dataset files with memory optimization"""  
 self.\_extract\_data()  
 self.\_transform\_features()  
 self.\_calculate\_similarity()

My feature engineering approach involves several steps. I began with text vectorization using CountVectorizer for genre processing, which transforms textual genre data into numerical features suitable for similarity calculations. I then process director information through custom tokenization, ensuring that director influences are properly weighted in the final recommendations. These features are combined into a unified feature matrix that captures the essential characteristics of each movie.

## 4. Results and Analysis

The performance of my recommendation system demonstrates significant achievements in both computational efficiency and recommendation quality. Through careful implementation of memory optimization techniques, my system processes the quality-filtered dataset of 26,107 movies with remarkable efficiency. This took about 3 minutes to load and process the data.

A screen shot of a computer

Description automatically generated

Response time analysis shows that my implementation generates recommendations in less than three seconds after data is prepared, even when processing the full dataset. This performance is achieved through careful optimization of similarity calculations. The system’s memory remains stable during operation, demonstrating the effectiveness of my memory optimization strategies. Null Hypothesis of: The scores from the baseline system (no ratings) are greater than or equal to the scores from the ratings-enhanced system was rejected.

## 5. Conclusions and Future Work

My implementation of the IMDB movie recommender system successfully addresses the key challenges of large-scale data processing and quality recommendation generation. The system demonstrates performance in processing efficiency, memory management, and recommendation quality. Through careful optimization techniques, I have created a system that provides meaningful, explainable recommendations while maintaining computational efficiency.

Looking forward, I have identified several promising areas for enhancement. The system’s architecture supports the potential integration of additional metadata sources, which could further improve recommendation quality. Implementation of advanced caching mechanisms could reduce response times for frequently requested recommendations. Additionally, the development of distributed processing capabilities would allow the system to scale to even larger datasets while maintaining performance.

## 6. Complications

The development of this system was not without challenges. The most significant issue I encountered was determining the best way to load the dataset. Initially, I attempted to load the dataset locally, but this caused my Jupyter Notebook to crash repeatedly. To overcome this, I tried using an Amazon Workspace; however, this approach presented additional challenges that were beyond my control. Troubleshooting the AWS environment was time-consuming, requiring multiple hours of assistance from support as I navigated an unfamiliar platform.

After several failed attempts with AWS, I shifted to using Kaggle. Fortunately, I found the exact dataset uploaded there. I retooled my code to integrate the Kaggle API for dataset retrieval. While I successfully loaded the data, it did not integrate seamlessly with my existing code. Debugging this issue took a considerable amount of time, as I spent half a day configuring the necessary adjustments to align with the new data source.

Finally, I sought a more efficient solution and opted to purchase the premium version of Google Colab. This version provided access to their V28 system, equipped with the resources required to process my dataset directly within the Colab environment. This transition marked the final iteration of the system you see today, and I was thrilled to finally have a stable and reliable platform for my project.

### References

1. IMDB Dataset Documentation (2024). “IMDB Datasets.” Retrieved from https://www.imdb.com/interfaces/